实验报告

LMS 自适应滤波器

王子龙

18281218 | 计科 1804 wangzilong@bjtu.edu.cn

目录

自适应滤波器

自适应滤波器是近 30 年来发展起来的关于信号处理方法和技术的滤波器，其设计方法对滤波器的性能影响很大。维纳滤波器等滤波器设计方法都是建立在信号特征先验知识基础上的。遗憾的是，在实际应用中常常无法得到信号特征先验知识，在这种情况下，自适应滤波器能够得到比较好的滤波性能。当输入信号的统计特性未知，或者输入信号的统计特性变化时，自适应滤波器能够自动地迭代调节自身的滤波器参数，以满足某种准则的要求，从而实现最优滤波。因此，自适应滤波器具有“自我调节”和“跟踪”能力。自适应滤波器可以分为线性自适应滤波器和非线性自适应滤波器。非线性自适应滤波器包括 Volterra 滤波器和基于神经网络的自适应滤波器。非线性自适应滤波器具有更强的信号处理能力。但是，由于非线性自适应滤波器的计算较复杂，实际用得最多的仍然是线性自适应滤波器[1]。本文只讨论线性自适应滤波器及其算法。图 1 为自适应滤波器原理框图。

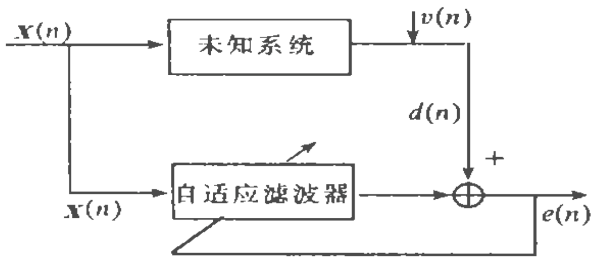

图 1 自适应滤波器原理图

自适应滤波算法广泛应用于系统辨识、回波消除、自适应谱线增强、自适应信道均衡、语音线性预测、自适应天线阵等诸多领域中。$W(n)$表示自适应滤波器在时刻n的权矢量，$x(n) = [x(n), x(n-1), \ldots, x(n-L+1)]^T$为时刻$n$的输入信号矢量，$d(n)$为期望输出值，$v(n)$为干扰信号，$e(n)$是误差信号，$L$是自适应滤波器的长度。最小均方差（LMS）是一种基本的自适应滤波器算法[2]。基于最小均方误差的准则，LMS 算法使滤波器的输出信号与期望输出信号之间的均方误差$E[e^2(n)]$最小。

LMS 自适应滤波器

1959 年，Widrow 和 Hof 在对自适应线性元素的方案—模式识别进行研究时，提出了最小均方算法（简称 LMS 算法）。LMS 算法具体如何推导的来的呢？它首先基于维纳滤波，然后借助于最速下降算法发展起来的。通过维纳滤波所求解的维纳解，必须在已知输入信号与期望信号的先验统计信息，以及再对输入信号的自相关矩阵进行求逆运算的情况下才能得以确定。因此，这个维纳解仅仅是理论上的一种最优解。所以，又借助于最速下降算法，以递归的方式来逼近这个维纳解，从而避免了矩阵的求逆运算，但仍然需要信号的先验信息，故而再使用瞬时误差的平方来代替均方误差，从而最终得出了 LMS 算法。

图 2 是实现 LMS 算法的一个矢量信号流程图：

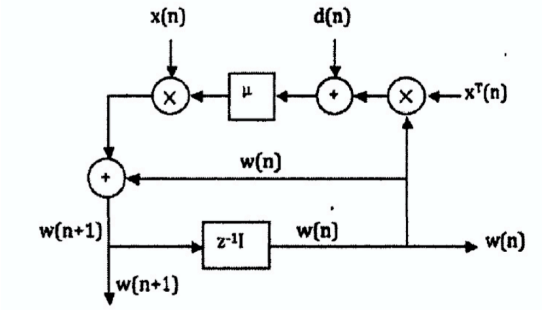

图 2 LMS 算法矢量信号流程图

由图 2 我们可以知道，LMS 算法主要包含两个过程：滤波处理和自适应调整。一般情况下，LMS 算法的具体流程为：

1） **确定参数**：全局步长参数β以及滤波器阶数 N
2） **对滤波器初始值的初始化**
3） **算法运算过程**：

滤波输出：$$y(n) = w^T(n)x(n) \quad (1)$$
误差信号：$$e(n) = d(n) - y(n) \quad (2)$$
权系数更新：$$w(n+1) = w(n) + \beta e(n)x(n) \quad (3)$$

牛顿法 LMS 自适应滤波器

LMS 算法是对均方误差准则$J(n) = E[e^2(n)]$的梯度矢量用瞬时估计值替代而得到的。虽然 LMS 算法因其结构简单，鲁棒性强而被广泛使用，但其收敛速度很慢。这是因为 LMS 算法的收敛速度依赖于输入信号自相关矩阵的特征值发散度。当输入信号的相关性越强，自相关矩阵的特征值发散度就越大，算法的收敛速度也就越慢。另外，LMS 算法中的步长因子μ对算法的收敛性能影响很大。小的μ值可以减少梯度噪声，但收敛速度慢；大的μ值可以加快收敛速度，但同时增加了梯度噪声和稳态失调；故其步长因子μ的选择难以把握。对于均方误差准则$J(n) = E[e^2(n)]$的梯度：

$$\nabla = \frac{\partial J_r(n)}{\partial w} = 2Rw - 2P$$

令其等于零得最佳权向量w_0满足 Wiener-Hopf 方程

$$w_0 = R^{-1}P$$

其中$R = E[x(n)x^T(n)]$，$P = E[d(n)x(n)]$。

由两式得$w_0 = w - \frac{1}{2}R^{-1}\nabla$，写为自适应迭代算法（即牛顿算法）形式：

$$w(n+1) = w(n) - \frac{1}{2}R^{-1}\nabla$$

式中n表示迭代步数，$\nabla(n)$为第n步迭代式权向量$w(n)$处的梯度。在实际应用中，用$\hat{\nabla}(n)$的估计值来替代$\nabla(n)$而得 LMS 牛顿算法如下：

$$w(n+1) = w(n) - \mu R^{-1}\hat{\nabla}(n)$$

这里$0 < \mu < 1$，应用收敛因子μ是为了保证$\nabla(n)$的噪化估计也能使算法收敛。LMS 牛顿算法解决了 LMS 算法收敛速度很慢的问题。

NLMS 自适应滤波器

NLMS 算法就是进行归一化后的 LMS 算法，又称归一化 LMS 算法，其实，NLMS 算法可以看作是一种特别的变步长 LMS 算法，常用于回声消除领域。在结构方面，NLMS 自适应滤波器和 LMS 自适应滤波器是相同的，都为横向滤波器，它们的不同只在于权值控制器方面，这是由两者算法中的系数更新项不同所引起的。其实，NLMS 算法可以看成是对原 LMS 算法的一种改进。

NLMS 算法的滤波器系数更新式的具体描述为：

$$w(n+1) = w(n) + \frac{\beta}{\|x(n)\|^2}x(n)e(n)$$

上式中，$\|x(n)\|^2$是输入信号$x(n)$的欧式范数平方。与传统的 LMS 算法的滤波器更新式$w(n+1) = w(n) + \beta e(n)x(n)$相比，NLMS 算法可以看成是一种变步长的 LMS 算法，所以，相比 LMS 算法，NLMS 算法将具有更好的收敛性能。当然，在收敛之前，NLMS 算法也必须要满足一定的收敛条件：

$$0 < \frac{\beta}{\|x(n)\|^2} < \frac{2}{tr[R]}$$

对上式近似化简的，$0 < \beta < 2$，也就是当步长β满足条件$0 < \beta < 2$时算法就会收敛，此时不再受输入信号特征值的影响。此时 NLMS 算法的滤波器系数更新式中若输入向量$x(n)$过小，则有可能导致算式发散。为了避免此类情况的产生，NLMS 算法应该改写为：

$$w(n+1) = w(n) + \frac{\beta}{\|x(n)\|^2 + \varepsilon}x(n)e(n)$$

其中，上式中$0 < \varepsilon < 1$。

对于 LMS 算法，当输入信号$x(n)$比较大时，其进入稳态后的噪声矢量公式为：

$$N(n) = -2e(n)x(n)$$

对于 NLMS 算法，它的稳态噪声矢量公式为：

$$N'(n) = -2e(n)\frac{x(n)}{\|x(n)\|^2}$$

对比两噪声矢量公式可以明显看出，与 LMS 算法相比，NLMS 算法的失调量将会减少（失调是由稳态后的噪声引起的），同时因输入向量$x(n)$过大而产生的噪声也会减小。此外，NLMS 算法还因变步长具有了更快的收敛速度，同时还与 LMS 算法的计算量相当。因此 NLMS 算法比 LMS 算法拥有着更为广泛的实际应用。

代码改进

修正的 LMS 牛顿法

LMS 牛顿法解决了 LMS 算法收敛速度很慢的缺点，但是其步长因子μ的选择仍然难以把握。根据[6]中提出的修正 LMS 算法的思想（即用当前时刻的梯度估计代替前一时刻的梯度估计）以及矩阵求逆定理导出了一种修正 LMS 算法。

算法推导

根据[6]中提出的利用现时刻的梯度估计代替前一时刻的梯度估计的思想，可以将 LMS 牛顿法中的式子$w(n+1) = w(n) - \mu R^{-1}\hat{\nabla}(n)$修正为

$$w(n+1) = w(n) - \mu R^{-1}\hat{\nabla}(n+1)$$

将其用瞬时梯度信息可表示为：

$$w(n+1) = w(n) - \mu R^{-1}e(n+1)x(n+1)$$

将$e(n+1) = d(n+1) - w^T(n+1)x(n+1)$带入上式整理后有：

$$[I + \mu R^{-1}x(n+1)x^T(n+1)]w(n+1) = [w(n) + \mu R^{-1}d(n+1)x(n+1)]$$

两边乘以矩阵R有：

$$[R + \mu x(n+1)x^T(n+1)]w(n+1) = [Rw(n) + \mu d(n+1)x(n+1)]$$

利用矩阵求逆定理[2]整理后带入上式后得：

$$w(n+1) = w(n) + \frac{\mu R^{-1}x(n+1)\overline{e}(n+1)}{1 + \mu x^T(n+1)R^{-1}x(n+1)}$$

其中$\overline{e}(n+1) = d(n+1) - w^T(n)x(n+1)$

一般R用其估计$\hat{R}(n)=\frac{1}{n+1}\sum_{i=0}^{n}x(i)x^T(i)=\frac{n}{n+1}\hat{R}(n-1)+\frac{1}{n+1}x(n)x^T(n)$来表示，再次利用矩阵求逆定理课的$\hat{R}^{-1}(n)$的递推估计式：

$$\hat{R}^{-1}(n)=\frac{n+1}{n}\hat{R}^{-1}(n-1)-\frac{\hat{R}^{-1}(n-1)x(n)x^T(n)\hat{R}^{-1}(n-1)}{x+x^T(n)\hat{R}^{-1}(n-1)x(n)}$$

算法核心

[5]中描述的修正 LMS 牛顿法由以下三式组成：

$$w(n+1)=w(n)+\frac{\mu R^{-1}x(n+1)\overline{e}(n+1)}{1+\mu x^T(n+1)R^{-1}x(n+1)}$$

$$\overline{e}(n+1)=d(n+1)-w^T(n)x(n+1)$$

$$\hat{R}^{-1}(n)=\frac{n+1}{n}\hat{R}^{-1}(n-1)-\frac{\hat{R}^{-1}(n-1)x(n)x^T(n)\hat{R}^{-1}(n-1)}{x+x^T(n)\hat{R}^{-1}(n-1)x(n)}$$

其中初始化条件应该为：$w(0)=0$，$\hat{R}^{-1}(0)=\delta I$，δ为小的正数，I为$L\times L$单位矩阵。

根据[5]中所给出的公式，我对西安交通大学 M. Zhang 的代码中的 LMS 库进行了改进，在其中加入了修正 LMS 牛顿法函数。具体实现内容在下文可见。

程序

程序源文件

模块名称	文件名称	文件说明
主模块	main.cpp	用于程序测试，入口程序
常量以及名字空间声明	constants.h	一些用于数值计算的常量
	usingdeclare.h	通常使用的名字空间声明
向量类	vector.h	vector 类被设计用来进行基础的向量线性代数运算
	vector-impl.h	vector 类的实现
矩阵类	matrix.h	matrix 类被设计用来进行基础的矩阵线性代数运算
	matrix-impl.h	matrix 类的实现
LMS 算法核心实现	lms.h	四种 LMS 算法的声明
	lms-impl.h	四种 LMS 算法的实现

lms-impl.h

lms-impl.h 内包含了四种 LMS 自适应滤波器，分别为常规的 LMS 自适应滤波器算法、牛顿法 LMS 自适应滤波器算法、修正牛顿法 LMS 自适应滤波器算法以及归一化的 LMS 自适应滤波器算法。

常规的 LMS 自适应滤波器

以下是常规 LMS 自适应滤波器的核心代码：

```
template <typename Type>
Type lms( const Type &xk, const Type &dk, Vector<Type> &wn,
const Type &mu )
{
    int filterLen = wn.size();
    static Vector<Type> xn(filterLen);
    // 更新输入信号
    for( int i=filterLen; i>1; --i )
        xn(i) = xn(i-1);
    xn(1) = xk;
    // 获取输出
    Type yk = dotProd( wn, xn );
    // 更新权重向量
    wn += 2*mu*(dk-yk) * xn;
    return yk;
}
```

牛顿 LMS 自适应滤波器

以下是牛顿 LMS 自适应滤波器的核心代码：

```
template <typename Type>
Type lmsNewton( const Type &xk, const Type &dk, Vector<Type>
&wn, const Type &mu, const Type &alpha, const Type &delta )
{
    assert( 0 < alpha );
    assert( alpha <= Type(0.1) );

    int filterLen = wn.size();
    Type beta = 1-alpha;
    Vector<Type> vP(filterLen);
    Vector<Type> vQ(filterLen);

    static Vector<Type> xn(filterLen);

    // 初始化相关矩阵的逆
    static Matrix<Type> invR = eye( filterLen,
Type(1.0/delta) );

    // 更新输入信号
    for( int i=filterLen; i>1; --i )
        xn(i) = xn(i-1);
```

```
    xn(1) = xk;

    Type yk = dotProd(wn,xn);

    // 更新相关矩阵的逆
    vQ = invR * xn;
    vP = vQ / (beta/alpha+dotProd(vQ,xn));
    invR = (invR - multTr(vQ,vP)) / beta;

    // 更新权重向量
    wn += 2*mu * (dk-yk) * (invR*xn);

    return yk;
}
```

新实现的改进后的牛顿法 LMS 自适应滤波器

以下是改进后的牛顿法 LMS 自适应滤波器的核心代码：

```
template <typename Type>
Type lmsNewtonFix(const Type &xk, const Type &dk, Vector<Type>
&wn, const Type &mu, const Type &alpha, const Type &delta) {
    int filterLen = wn.size();

    static Vector<Type> xn(filterLen);
    static Vector<Type> I(filterLen);
    for (int i = 0; i < filterLen; ++i) {
        I[i] = 1;
    }

    // 初始化相关矩阵的逆
    static Matrix<Type> invR = eye( filterLen,
Type(1.0*delta) );

    // 更新输入信号
    for( int i=filterLen; i>1; --i )
        xn(i) = xn(i-1);
    xn(1) = xk;

    // 更新相关矩阵的逆
    invR = (filterLen + 1.0) / (filterLen * 1.0)  * invR - (invR
* dotProd(xn ,xn) * invR)/(xk + sDotProd(xn, invR) * xk);
    Type yk = dotProd(wn,xn);
    // 更新权向量
    wn += (mu * invR * xn * (dk - dotProd(wn, xn)))/(I + mu *
```

```
    invR * sDotProd(xn, invR) * xn);

        return yk;
    }
```

归一化 LMS 自适应滤波器

以下是归一化 LMS 自适应滤波器的核心代码：

```
template <typename Type>
Type lmsNormalize( const Type &xk, const Type &dk, Vector<Type>
&wn, const Type &rho, const Type &gamma )
{
    assert( 0 < rho );
    assert( rho < 2 );

    int filterLen = wn.size();
    static Vector<Type> sn(filterLen);

    // 更新输入信号
    for( int i=filterLen; i>1; --i )
        sn(i) = sn(i-1);
    sn(1) = xk;

    // 获取输出
    Type yk = dotProd( wn, sn );

    // 更新权重向量
    wn += rho*(dk-yk)/(gamma+dotProd(sn,sn)) * sn;

    return yk;
}
```

遇到的问题

在完成改进后的牛顿法 LMS 自适应滤波器部分代码时，因为算法中存在 $x^T(n)\hat{R}^{-1}(n-1)x(n)$这样的式子，因此需要进行向量点乘矩阵的运算。而在西安交大的 Zhang Ming 实现的算法中，并没有实现相应算法，因此我对向量与矩阵的点乘（结果为数字）进行了实现：

```
template<typename Type>
Type sDotProd(Vector<Type>&v, Matrix<Type>&m) {
    Type ans = 0;
    int n = v.size();
```

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            ans += v[j] * m[j][i];
        }
    }

    return ans;
}
```

实验

实验环境

操作系统： macOS Catalina 10.15.5 (19F101)
算法语言： C++
编译环境： Apple clang version 11.0.0 (clang-1100.0.33.8)
集成环境： CLion 2020.1.2(201.7846.488)

实验过程

首先生成信号，并分别通过四种 LMS 自适应滤波器进行滤波并得到结果。

实验结果（正确性）及性能分析

The last 10 iterations of Conventional-LMS:

observed	desired	output	adaptive filter
-0.2225	-0.9749	-0.8216	-0.5683 1.0810
0.6235	-0.7818	-0.5949	-0.5912 1.0892
1.0000	-0.0000	0.0879	-0.6084 1.0784
0.6235	0.7818	0.6991	-0.5983 1.0947
-0.2225	0.9749	0.8156	-0.6053 1.1141
-0.9010	0.4339	0.2974	-0.6294 1.1082
-0.9010	-0.4339	-0.4314	-0.6289 1.1086
-0.2225	-0.9749	-0.8589	-0.6239 1.1291
0.6235	-0.7818	-0.6402	-0.6412 1.1353
1.0000	-0.0000	0.0667	-0.6543 1.1271

The last 10 iterations of LMS-Newton:

observed	desired	output	adaptive filter
-0.2225	-0.9749	-0.9739	-0.7958 1.2779
0.6235	-0.7818	-0.7805	-0.7964 1.2783

1.0000	-0.0000	0.0006	-0.7966	1.2783
0.6235	0.7818	0.7816	-0.7966	1.2784
-0.2225	0.9749	0.9743	-0.7968	1.2787
-0.9010	0.4339	0.4334	-0.7971	1.2787
-0.9010	-0.4339	-0.4340	-0.7971	1.2787
-0.2225	-0.9749	-0.9747	-0.7971	1.2788
0.6235	-0.7818	-0.7816	-0.7972	1.2789
1.0000	-0.0000	0.0001	-0.7973	1.2789

The last 10 iterations of LMS-Newton-Fix:

observed	desired	output	adaptive filter	
-0.2225	-0.9749	-0.9719	-0.7987	1.2757
0.6235	-0.7818	-0.7819	-0.7987	1.2757
1.0000	-0.0000	-0.0033	-0.7984	1.2761
0.6235	0.7818	0.7783	-0.7972	1.2773
-0.2225	0.9749	0.9738	-0.7965	1.2778
-0.9010	0.4339	0.4333	-0.7967	1.2777
-0.9010	-0.4339	-0.4333	-0.7964	1.2779
-0.2225	-0.9749	-0.9742	-0.7962	1.2782
0.6235	-0.7818	-0.7809	-0.7971	1.2793
1.0000	-0.0000	0.0006	-0.7973	1.2791

The last 10 iterations of Normalized-LMS:

observed	desired	output	adaptive filter	
-0.2225	-0.9749	-0.9749	-0.7975	1.2790
0.6235	-0.7818	-0.7818	-0.7975	1.2790
1.0000	-0.0000	-0.0000	-0.7975	1.2790
0.6235	0.7818	0.7818	-0.7975	1.2790
-0.2225	0.9749	0.9749	-0.7975	1.2790
-0.9010	0.4339	0.4339	-0.7975	1.2790
-0.9010	-0.4339	-0.4339	-0.7975	1.2790
-0.2225	-0.9749	-0.9749	-0.7975	1.2790
0.6235	-0.7818	-0.7818	-0.7975	1.2790
1.0000	-0.0000	-0.0000	-0.7975	1.2790

The theoretical optimal filter is: -0.7972 1.2788

可以看出，所有算法均能正确运行。在几种 LMS 自适应滤波器中，最优的是归一化的 LMS 自适应滤波。其次是牛顿法 LMS 自适应滤波器以及修正的牛顿法 LMS 自适应滤波器。效果最差的是基础的 LMS 自适应滤波器。在调参数时，LMS 滤波器的效果与其步长因子μ的选择有着极大关系，通常来说，一般滤波器的阶数越大，它的取值越小。LMS 算法易于实现、性能稳定、应用广泛，而且运算量很小，通常为$O(n)$数量级。

总结

自适应滤波理论是现代统计信号处理的重要基础,近年来发展相当迅速，在实现自适应滤波的众多算法中 LMS 算法凭借其简单的结构设计、较低的计算复杂度等优点被人们广泛研究和使用。LMS 自适应滤波广泛应用于系统辨识、语音预测、回波消除、自适应信道均衡、天线波束形成、噪声消除等领域[7]。对于这些不同的应用，基本原理都是相同的，只是所加的输入信号和期望信号不同。本文基于西安交通大学 Zhang Ming 的 LMS 自适应滤波代码以及高鹰的 “LMS 牛顿自适应滤波算法的一种修正形式” 完成了四种不同 LMS 自适应滤波器代码的编写并进行了实验。实际上针对 LMS 自适应滤波算法，有着性能更优，效率更高的改进算法。比如各种变步长的 LMS 自适应滤波算法，可以很好地提升性能。如蒋明峰等人提出的一种基于误差自相关估计的 MVSS-LMS 自适应滤波算法[8]。但由于时间等原因，在本报告中未能对更多算法进行实现。

参考文献

[1]. 邹艳碧，高鹰，自适应滤波算法综述[J]，广州大学学报，2002 年 2 月，Vol.1 No.2.
[2]. S Haykin .Adaptive filtering theory (4th Edition)[M] .Prentice_Hall, 2002.
[3]. A.H.Sayed. Fundamentals of Adaptive Filtering. New York, Wiley, NJ, 2003.
[4]. B.Farhang-Boroujeny. Adaptive Filters: Theory and Applications. New York, Wiley, 1998.
[5]. 高鹰，谢胜利，LMS 牛顿自适应滤波算法的一种修正形式，第二十二届中国控制会议，2003.
[6]. F,F,Jretschmer, Jr, B.L.Lewis, An improved algorithm for adaptive processing, IEEE Trans. 1978.
[7] 李宁，LMS 自适应滤波算法的收敛性能研究及应用[博士学位论文]，哈尔滨工程大学，2009.
[8] 蒋明峰，郑小林，彭承琳. 一种新的变步长 LMS 自适应算法及其在自适应噪声对消中的应用[J]. 信号处理, 2001, 27: 282-256.

附录

代码地址：http://github.com/EricWangCN/LMS